Redes Neurais Artificiais

Dupla: Larissa Sousa, Ruan Rodrigues

PROJETO FINAL

O conjunto de dados Breast Cancer Wisconsin (Diagnostic) é uma valiosa fonte de informações para a detecção de câncer de mama. Coletado na Universidade de Wisconsin, o conjunto oferece características derivadas de biópsias, como raio médio, textura média e concavidade média. Com o objetivo de prever diagnósticos, o conjunto categoriza tumores como malignos (M) ou benignos (B). Este estudo visa aplicar técnicas de aprendizado de máquina, especificamente um modelo de rede neural, para aprimorar a classificação dos tumores. A distinção precisa entre maligno e benigno é crucial para o diagnóstico precoce e tratamento eficaz do câncer de mama, tornando este conjunto de dados fundamental para avanços na área médica. O conjunto de dados contém diversas características computadas a partir de imagens digitalizadas de aspirados de agulha fina (FNA) de tumores de mama. As características incluem medidas como raio, textura, perímetro, área, suavidade, compacidade, concavidade, pontos côncavos, simetria e dimensão fractal.

O objetivo típico ao trabalhar com este conjunto de dados é desenvolver modelos de aprendizado de máquina capazes de classificar tumores de mama como benignos ou malignos com base nessas características.

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
pd.set_option('display.max_columns',None)
sns.set_style('darkgrid')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import ReduceLROnPlateau
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
```

```
df=pd.read_csv(r'/content/data.cancer.csv')
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothr
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

```
colunas = df.columns
print(len(colunas), "colunas")
```

```
33 colunas
```

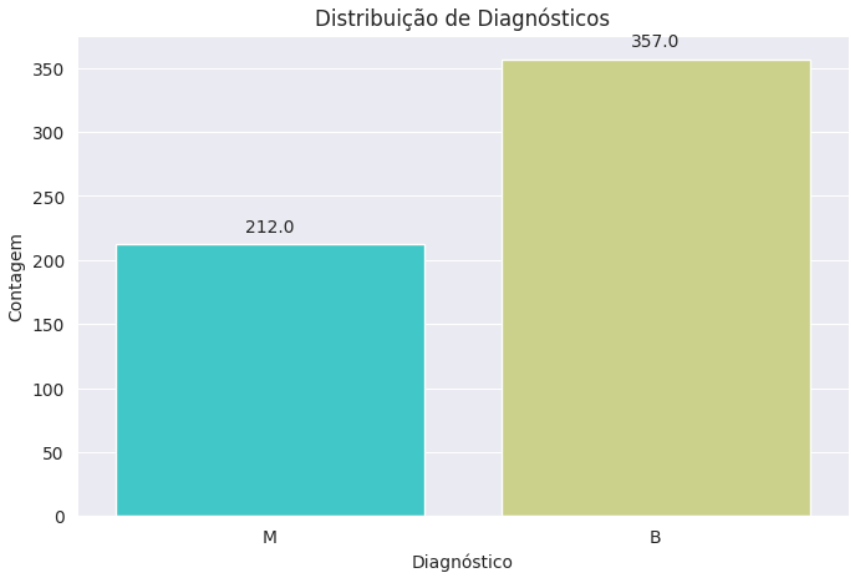
```
df.drop(['id','Unnamed: 32'],axis=1,inplace=True)
contagem_diagnosis = df['diagnosis'].value_counts()
print(contagem_diagnosis)
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 5))
ax = sns.countplot(x='diagnosis', data=df, palette='rainbow')

for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 10), textcoords='offset points')
plt.xlabel('Diagnóstico')
plt.ylabel('Contagem')
plt.title('Distribuição de Diagnósticos')
plt.show()
```



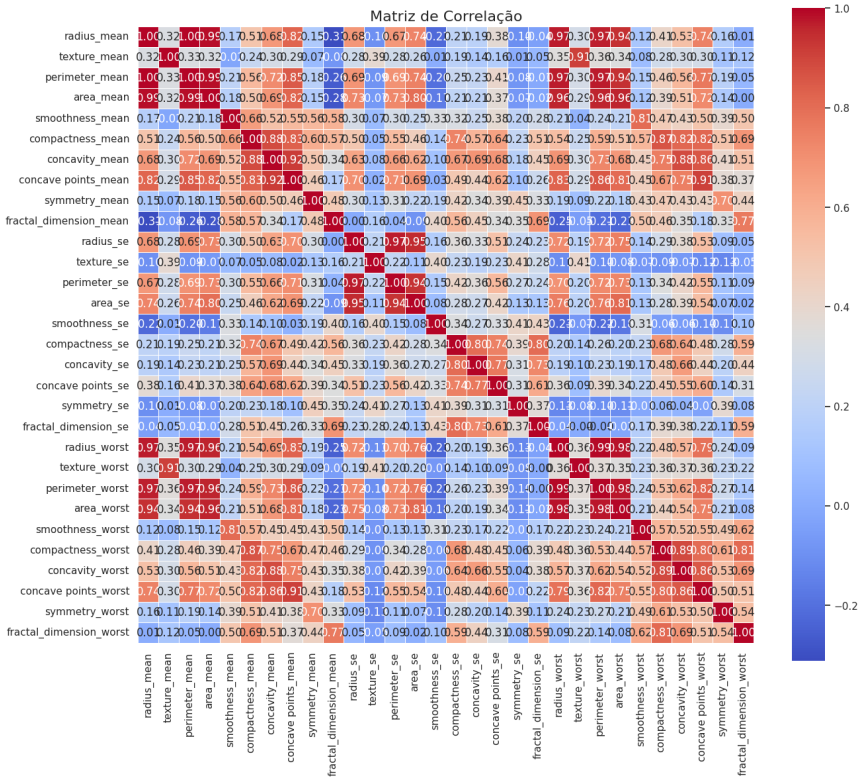

```
import seaborn as sns
import matplotlib.pyplot as plt

correlation_matrix = df.corr()

sns.set(style="white")

plt.figure(figsize=(14, 12))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5, square=True)
plt.title("Matriz de Correlação", fontsize=16)
plt.tight_layout()

plt.show()
```



Construindo um modelo sequencial de rede neural com camadas densas, utilizando a função de ativação ReLU nas camadas internas e sigmoid na camada de saída para classificação binária. A função de perda utilizada é 'binary_crossentropy' e o otimizador é 'Adam'.

Essa abordagem segue um processo típico para problemas de classificação binária usando redes neurais. O modelo resultante pode ser treinado e avaliado para ver como ele se sai na tarefa específica de classificar tumores como malignos ou benignos com base nas características fornecidas.

```
#separating target and independent features
y=df['diagnosis']
X=df.drop('diagnosis',axis=1)
#scaling the dataset
scaler_=StandardScaler()
X=pd.DataFrame(scaler_.fit_transform(X),columns=X.columns)
#splitting the data into train and test set
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
#model building
model=Sequential()
model.add(Dense(32,activation='relu',input_dim=X_train.shape[1]))
model.add(Dropout(0.5))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(8,activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='Adam',metrics=['accuracy'],loss='binary_crossentropy')
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                992

 dropout (Dropout)           (None, 32)                0

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 8)                 136

 dense_4 (Dense)             (None, 1)                 9

=================================================================
```

```
Total params: 2721 (10.63 KB)
Trainable params: 2721 (10.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.1,
                                            min_lr=0.000001)
history=model.fit(X_train, y_train, epochs=50, batch_size=32,shuffle=True,validation_split=0.2,callbacks=[learning_rate_reduction])
```

```
Epoch 1/50
---------------------------------------------------------------------------
UnimplementedError                        Traceback (most recent call last)
<ipython-input-10-35848df8b320> in <cell line: 6>()
      4                                             factor=0.1,
      5                                             min_lr=0.000001)
----> 6 history=model.fit(X_train, y_train, epochs=50, batch_size=32,shuffle=True,validation_split=0.2,callbacks=
[learning_rate_reduction])

 1 frames
/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs,
ctx, name)
     58         for t in inputs
     59     ]
---> 60     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     61                                         inputs, attrs, num_outputs)
     62   except core._NotOkStatusException as e:

UnimplementedError: Graph execution error:

Detected at node binary_crossentropy/Cast defined at (most recent call last):
  File "/usr/lib/python3.10/runpy.py", line 196, in _run_module_as_main

  File "/usr/lib/python3.10/runpy.py", line 86, in _run_code

  File "/usr/local/lib/python3.10/dist-packages/colab_kernel_launcher.py", line 37, in <module>

  File "/usr/local/lib/python3.10/dist-packages/traitlets/config/application.py", line 992, in launch_instance

  File "/usr/local/lib/python3.10/dist-packages/ipykernel/kernelapp.py", line 619, in start

  File "/usr/local/lib/python3.10/dist-packages/tornado/platform/asyncio.py", line 195, in start

  File "/usr/lib/python3.10/asyncio/base_events.py", line 603, in run_forever

  File "/usr/lib/python3.10/asyncio/base_events.py", line 1909, in _run_once

  File "/usr/lib/python3.10/asyncio/events.py", line 80, in _run
```

ANALISANDO MODELO

```
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(14,8))
epochs_=np.arange(50)
ax[0].plot(epochs_,history.history['accuracy'],'-o',color='black',label='train_acc')
ax[0].plot(epochs_,history.history['val_accuracy'],'-o',color='red',label='val_acc')
ax[1].plot(epochs_ , history.history['loss'] , '-o' ,color='black',label='train_loss')
ax[1].plot(epochs_ ,history.history['val_loss'] , 'r-o' ,color='red',label='val_loss')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-0aa546155777> in <cell line: 3>()
      1 fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(14,8))
      2 epochs_=np.arange(50)
----> 3 ax[0].plot(epochs_,history.history['accuracy'],'-o',color='black',label='train_acc')
      4 ax[0].plot(epochs_,history.history['val_accuracy'],'-o',color='red',label='val_acc')
      5 ax[1].plot(epochs_ , history.history['loss'] , '-o' ,color='black',label='train_loss')

NameError: name 'history' is not defined
```

PESQUISAR NO STACK OVERFLOW

```
y_pred=model.predict(X_test)
y_pred=y_pred>0.5
print('Accuracy:{}'.format(accuracy_score(y_test,y_pred)))
print('F1_score:{}'.format(f1_score(y_test,y_pred)))
```

```
print(classification_report(y_test,y_pred))
```

Essa representação tabular mostra como o modelo está classificando corretamente e incorretamente as instâncias de tumores benignos e malignos no conjunto de teste. Em resumo, o modelo está apresentando um desempenho muito bom, com poucos casos de classificações incorretas.

```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score

# Calculando a matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)

# Calculando a acurácia e F1-score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Imprimindo os resultados
print("Matriz de Confusão:")
print(conf_matrix)

print("\nAcurácia: {:.2f}".format(accuracy))
print("F1-Score: {:.2f}".format(f1))

from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

# Calculando a matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)

# Calculando a acurácia e F1-score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Plotando a matriz de confusão
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Benigno', 'Maligno'], yticklabels=['Benigno', 'Maligno'])
plt.xlabel('Previsão')
plt.ylabel('Real')
plt.title('Matriz de Confusão RN\nAcurácia: {:.2f} | F1-Score: {:.2f}'.format(accuracy, f1))
plt.show()
```

```
y_train_pred = model.predict(X_train)
y_train_pred = y_train_pred > 0.5

print('Train Accuracy: {}'.format(accuracy_score(y_train, y_train_pred)))
print('Train F1_score: {}'.format(f1_score(y_train, y_train_pred)))
```

```
print(classification_report(y_train, y_train_pred))
```

Construção do Modelo de Random Forest:

Criado um modelo de Random Forest utilizando a classe RandomForestClassifier do scikit-learn, com 100 estimadores (árvores de decisão) e uma semente `(random_state)` para reprodutibilidade.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Supondo que você já carregou e pré-processou o conjunto de dados

# Substitua 'seu_dataframe' pelo nome real do seu DataFrame e 'Diagnostico' pelo nome da coluna alvo
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Dividindo o conjunto de dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criando o modelo de Random Forest
random_forest_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Treinando o modelo
random_forest_model.fit(X_train, y_train)

# Fazendo previsões
y_pred = random_forest_model.predict(X_test)

# Obtendo a matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualizando a matriz de confusão como um heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Benigno', 'Maligno'], yticklabels=['Benigno', 'Maligno'])
plt.xlabel('Previsto')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()

# Avaliando o modelo
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Exibindo os resultados
print(f'Acurácia: {accuracy}')
print(f'Relatório de Classificação:\n{classification_rep}')
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Supondo que você já carregou e pré-processou o conjunto de dados
```

KNN

```
X = df.drop('diagnosis', axis=1)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Substitua 'seu_dataframe' pelo nome real do seu DataFrame e 'Diagnosis' pelo nome da coluna alvo
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Dividindo o conjunto de dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criando o modelo de KNN
knn_model = KNeighborsClassifier(n_neighbors=5)

# Treinando o modelo
knn_model.fit(X_train, y_train)

# Fazendo previsões
y_pred = knn_model.predict(X_test)

# Avaliando o modelo
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Visualizando a matriz de confusão como um heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Benigno', 'Maligno'], yticklabels=['Benigno', 'Maligno'])
plt.xlabel('Previsto')
plt.ylabel('Real')
plt.title('Matriz de Confusão para KNN')
plt.show()

# Imprimindo métricas de avaliação
classification_rep = classification_report(y_test, y_pred)
print(f'Acurácia: {accuracy}')
print(f'Relatório de Classificação:\n{classification_rep}')
```